Tuning a High-Performance Math Library for AMD64 Technology

Tim Wilkens Ph.D.
Member of Technical Staff

Acknowledgments

Work carried out in collaboration with the **N**umerical **A**lgorithms **G**roup (**NAG**):

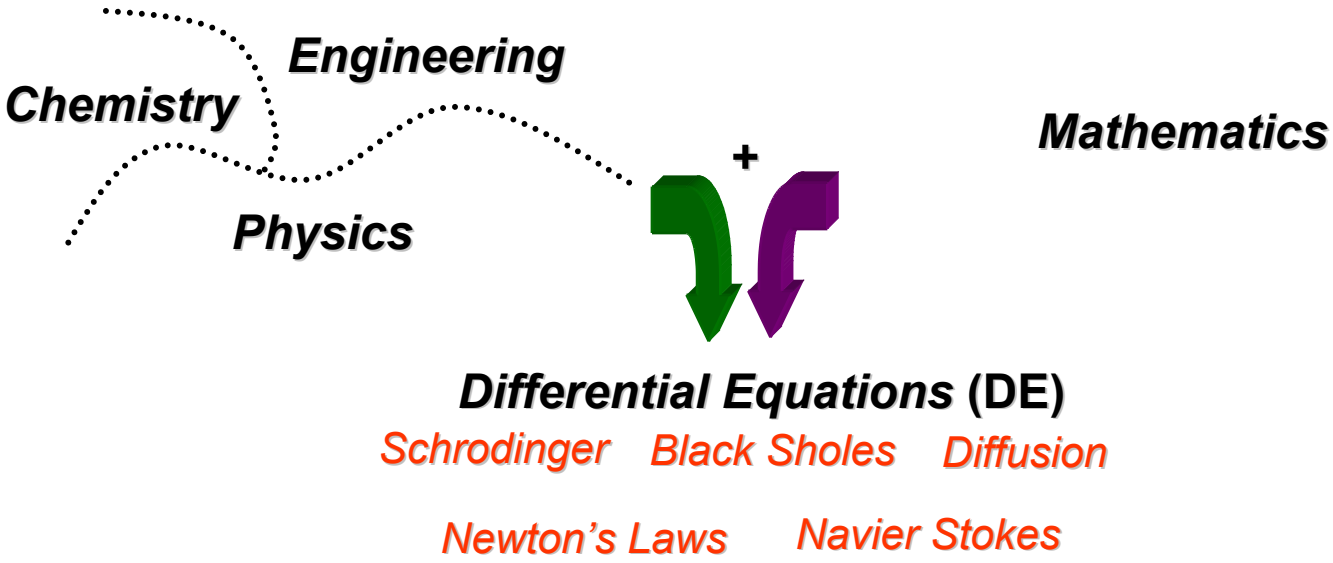
Lawrence Mulholland
Mick Pont
Stef Salvini
Ed Smyth
Themos Tsikas

NAG also provides its numerical libraries for AMD64™ which leverage ACML.

Agenda

- Motivation/Introduction to ACML
- Overview of Opteron Processor
- Optimization tactics employed
- ACML performance
- Future Improvements
- Summary and Closing Points

What are Math Libraries Good For?
Continuous vs Discrete



- **DE** equations in Calculus exist in continuous space
  - Solutions exist only for a small subset of **DE**

However, Transforming from Continuous to Discrete Space…

…all* DE *become Matrix Equations which can be solved using Linear Algebra

AMD Core Math Library (ACML)
Components

- **Linear Algebra (LA)**
  - **Basic Linear Algebra Subroutines (BLAS)**
    - Level 1 (vector-vector operations)
    - Level 2 (matrix-vector operations)
    - Level 3 (matrix-matrix operations)
    - Routines involving sparse vectors
  - **Linear Algebra PACKage (LAPACK)**
    - leverage BLAS to perform complex operations
    - 28 Threaded LAPACK routines
- **Fast Fourier Transforms (FFTs)**
  - 1D, 2D, 3D, single, double, r-r, r-c, c-r, c-c tranforms supported
- **C and Fortran interfaces**

AMD Core Math Library (ACML)
LAPACK

- Discrete Linear Differential Equations
- Least Squares Minimization problems
- Eigenvalue / Eigenvector problems
  - natural modes and frequencies
- Singular Value Decomposition
  - Condition # of problem
- Boundary Value Problems
  - Finite Element Method

ACML LAPACK leverages **Level 3 BLAS** routines,
performance gated by **FPU throughput *not* memory bandwidth**

AMD Core Math Library (ACML)
Where can it be used?

- **BLAS** and **LAPACK** uses
  - **ISV** specific routines (ex: **NASTRAN, GAUSSIAN**)
  - Chemistry **/** Life Science (ex: **GAMESS, AMBER**)
  - Sparse Solvers (*Boeing Spare Matrix Solver*)
  - University / Government Research
  - High Performance Linpack (**HPL**)
  - Business Intelligence (optimal solutions minimizing a set of constraints)

- **FFTs** uses
  - Games (adding realism to water surfaces)
  - Chemistry **/** Life Science (ex: Plane wave codes and **DFT**)
  - Digital Signal Analysis (who's voice is this?)
  - University / Government Research

AMD Core Math Library (ACML)
Assembly and Accuracy

Floating-point Numbers are represented in many formats specified by a set of parameters:

$$x = \pm \underbrace{\left(d_0 + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \ldots + \frac{d_{t-1}}{\beta^{t-1}} \right)}_{mantissa} \underbrace{\beta^e}_{exponent}$$

$$0 \le d_i \le \beta - 1 \qquad -N + 1 \le e \le N$$

$$i = 0, \ldots, t-1$$

The IEEE standard sets β to 2 to for beneficial reasons

t and ***N*** are dictated by the precision of the number being represented

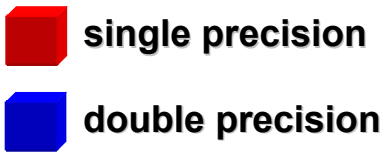
FP Operation	***t***	***N***
SSE	24	127
SSE2	53	1,023
X87	64	32,767

AMD Core Math Library (ACML)

Assembly usage in ACML

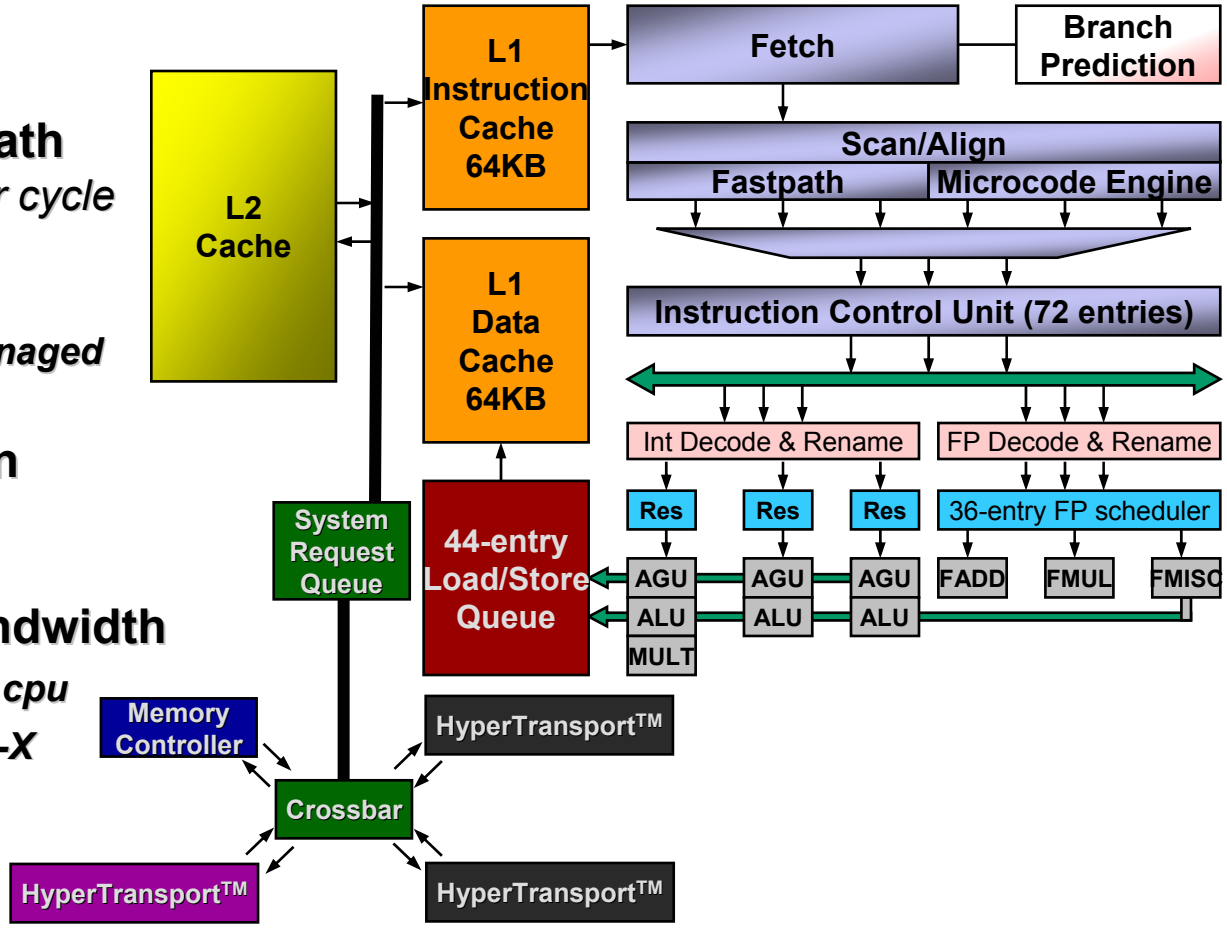
- 3 **ACML** 32-bit binaries
  - supports present and past AMD processors with/without SSE & SSE2
- 1 **ACML** 64-bit binary
  - supports AMD Athlon 64 and AMD Opteron

Address Space Supported	Processors Supported	BLAS Assembly Support	FFT Assembly Support
32-bit	AMD Athlon™ Processor	X87 X87	X87 X87
32-bit	AMD Athlon MP and XP Processor	SSE X87	SSE X87
32-bit	AMD Opteron™ Processor	SSE X87	SSE SSE2
64-bit	AMD Opteron Processor	SSE SSE2	SSE SSE2



The AMD Opteron™ Processor core
The Engine beneath the hood

- **RISC (not CISC) core**
  High Clock Frequency
- **DirectPath vs VectorPath**
  Execute 3 operations per cycle
- **Register Renaming**
  processor vs compiler managed
- **Out of Order Execution**
  Less code tuning required
- ***up to* 24.5 GB/s of Bandwidth**
  - *up to* ***12.8 GB/s cpu ⟺ cpu***
  - *up to* ***6.4 GB/s AGP, PCI-X***
  - *up to* ***5.3 GB/s Memory***

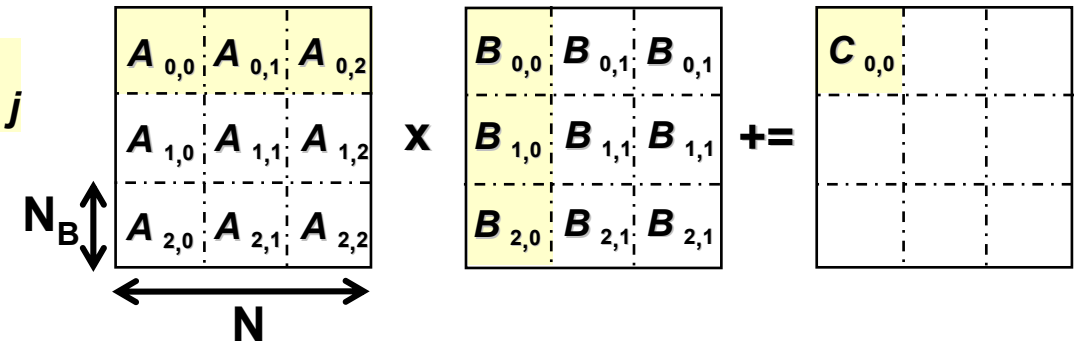


BLAS Optimization Tactics
Cache Blocking

- Generic **D**ouble precision **GE**neral **M**atrix-**M**ultiply (DGEMM) is performance limited by memory bandwidth for large problems

- Alternatively Block A and B into portions that fit into L1 cache and calculate blocks of C in the manner below:

$$C_{i,j} = \sum_{k=0}^{N/N_B} A_{i,k} \times B_{k,j}$$



- *Advantages*:
  - blocked data 16-byte aligned and local to CPU via NUMA
  - lots of L1 and L2 cache reusage

Provides approximately linear performance scalaring with CPU frequency and FPU throughput

BLAS Optimization Tactics
Prefetching

- Cache blocking promotes L2 and L1 data reusage *but* :
  - first access is still from memory
  - stalls FPU from keeping pipeline fully utilized

- Sum of block products ($A_{i,k} \times B_{k,j}$) is a predictable pattern
  - Interleave prefetch instructions into ($A_{i,k} \times B_{k,j}$) operation

- while operating $A_{i,k}$ upon column n of $B_{k,j}$ prefetch:
  - column $n+1$ of $B_{k,j}$
  - row $n+1$ of $A_{i,k+1}$
  - $C_{i,j}$ elements from L2 cache

Loads of A, B, and C blocks can be arranged to occur from the L1 cache → FPU efficiency can approach 100%

BLAS Optimization Tactics
Register Renaming in the Kernel

- Addresses can be large, no penalty to using Offsets larger than 8-bits (i.e. -128 : 127)

```
movapd (%rdi),%xmm1
movapd 256(%rdi),%xmm1
```

- **B** elements not reloaded. **A** elements loaded into **xmm1** out of order via register renaming. (**Using Hidden registers**)

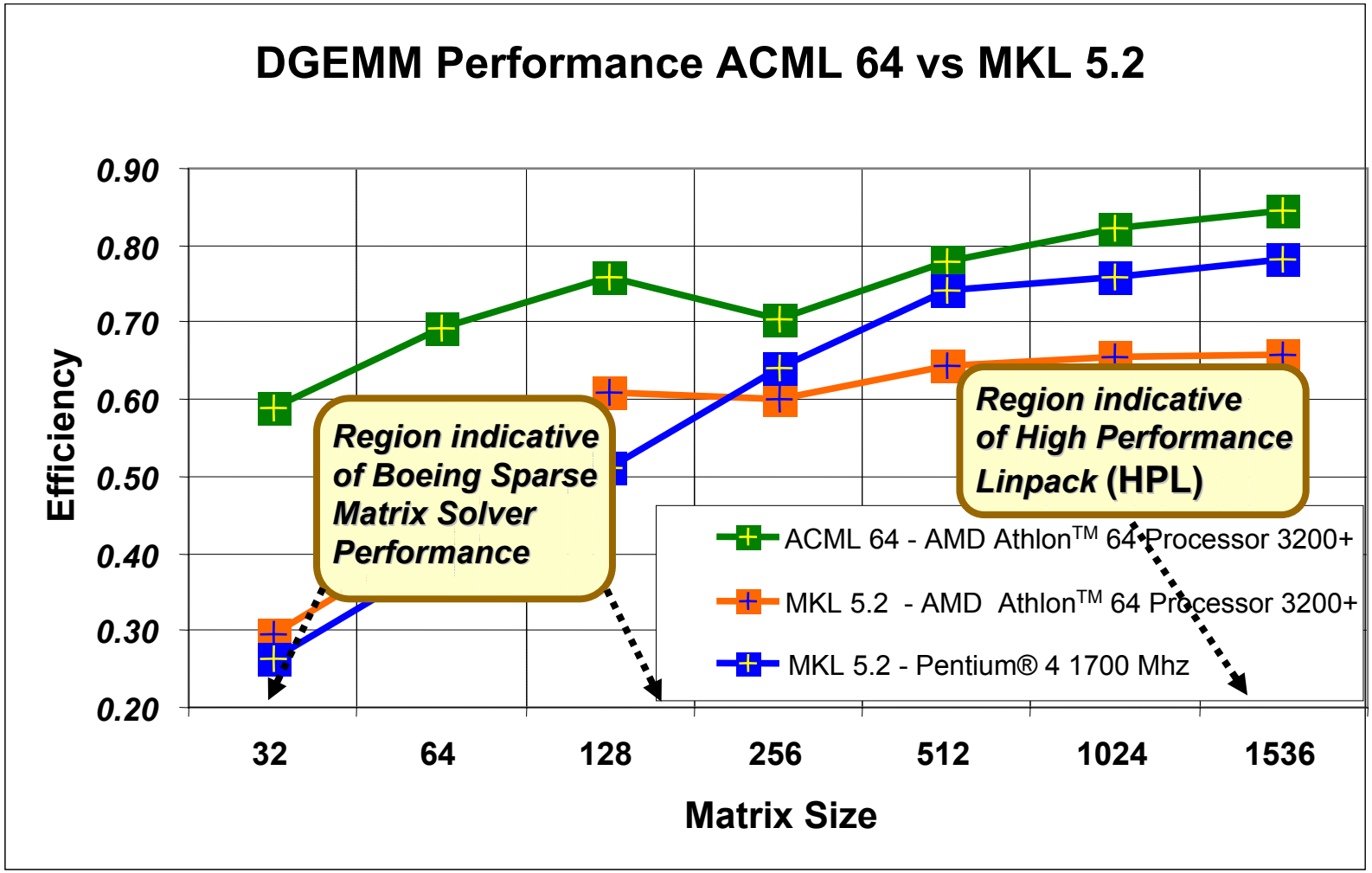
- **Load ops generate bubbles** which **can go down any of 3 FPU pipes**. Cases with *more than a (1:2) ratio* with executable ops *interfere with the mul/add* pipes. **Alleviated by multiplying $[A_{71},A_{70}]$ upon $[B_1,B_0]$ from memory**

```
movapd [B1,B0],%xmm0
movapd [A01,A00],%xmm1
mulpd %xmm0,%xmm1
addpd %xmm1,%xmm2
movapd [A11,A10],%xmm1
mulpd %xmm0,%xmm1
addpd %xmm1,%xmm3
movapd [A21,A20],%xmm1
mulpd %xmm0,%xmm1
addpd %xmm1,%xmm4
movapd [A31,A30],%xmm1
mulpd %xmm0,%xmm1
addpd %xmm1,%xmm5
movapd [A41,A40],% xmm1
mulpd %xmm0,%xmm1
addpd %xmm1,%xmm6
movapd [A51,A50],% xmm1
mulpd %xmm0,%xmm1
addpd %xmm1,%xmm7
movapd [A61,A60],% xmm1
mulpd %xmm0,%xmm1
addpd %xmm1,%xmm8
mulpd [A71,A70],% xmm0
addpd %xmm0,%xmm9
```

```
movapd [B3,B2],%xmm0
movapd [A03,A02],%xmm1
mulpd %xmm0,%xmm1
addpd %xmm1,%xmm2
movapd [A13,A12],%xmm1
mulpd %xmm0,%xmm1
addpd %xmm1,%xmm3
movapd [A23,A22],%xmm1
mulpd %xmm0,%xmm1
addpd %xmm1,%xmm4
movapd [A33,A32],%xmm1
mulpd %xmm0,%xmm1
addpd %xmm1,%xmm5
movapd [A43,A42],% xmm1
mulpd %xmm0,%xmm1
addpd %xmm1,%xmm6
movapd [A53,A52],% xmm1
mulpd %xmm0,%xmm1
addpd %xmm1,%xmm7
movapd [A63,A62],% xmm1
mulpd %xmm0,%xmm1
addpd %xmm1,%xmm8
mulpd [A73,A72],% xmm0
addpd %xmm0,%xmm9
```

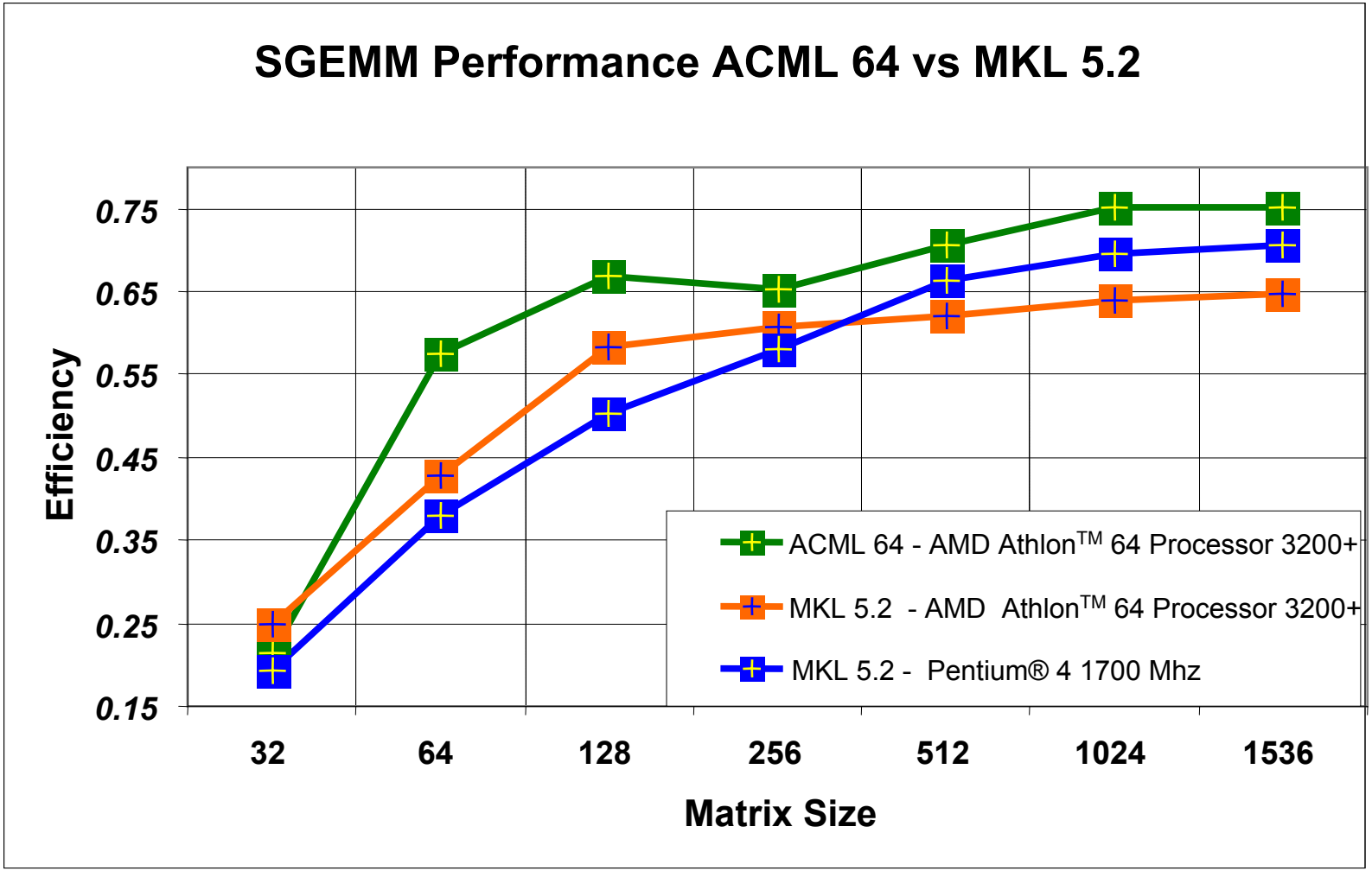
64-bit ACML v1.5 BLAS Performance

DGEMM



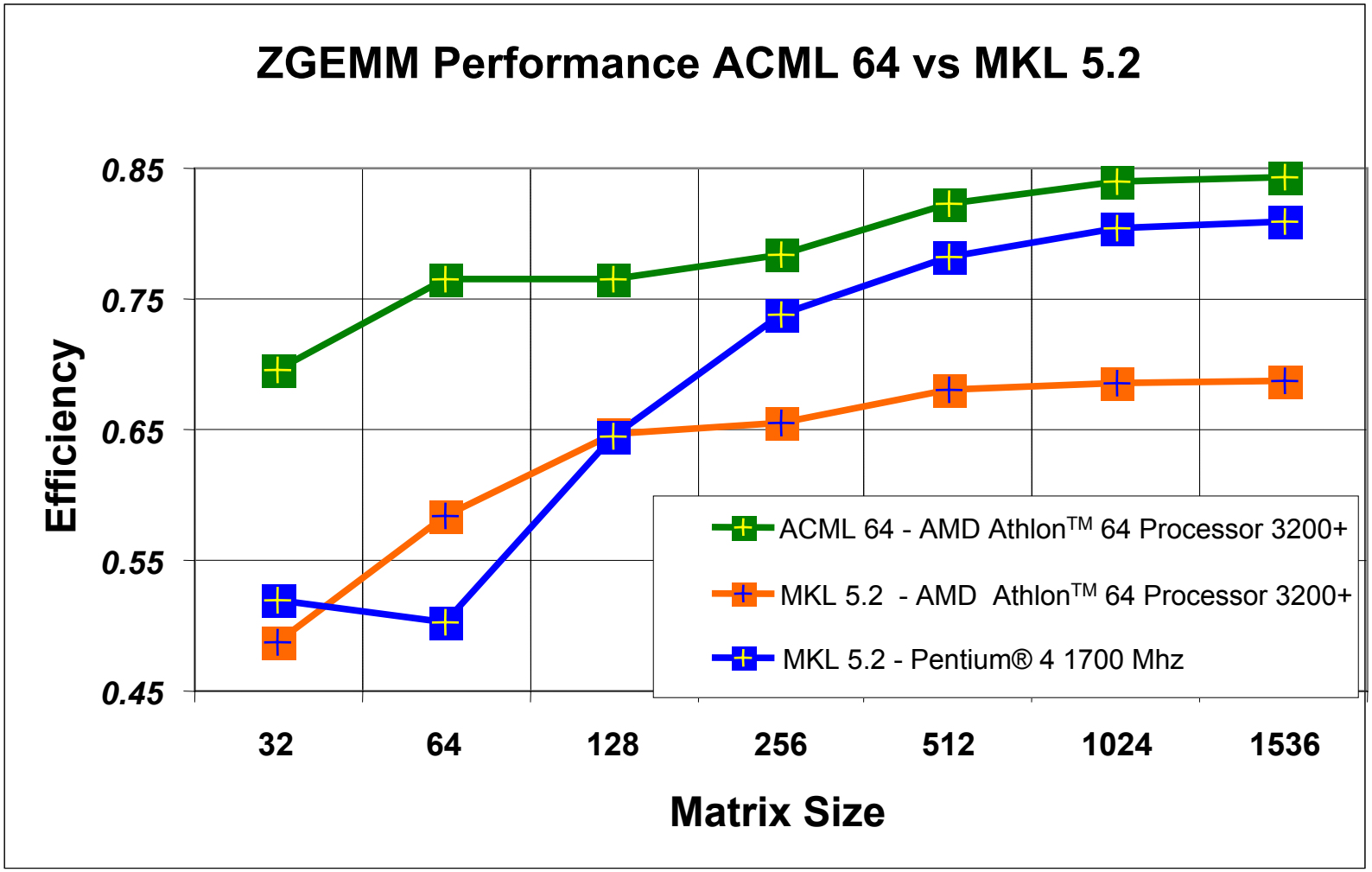
64-bit ACML v1.5 BLAS Performance

SGEMM



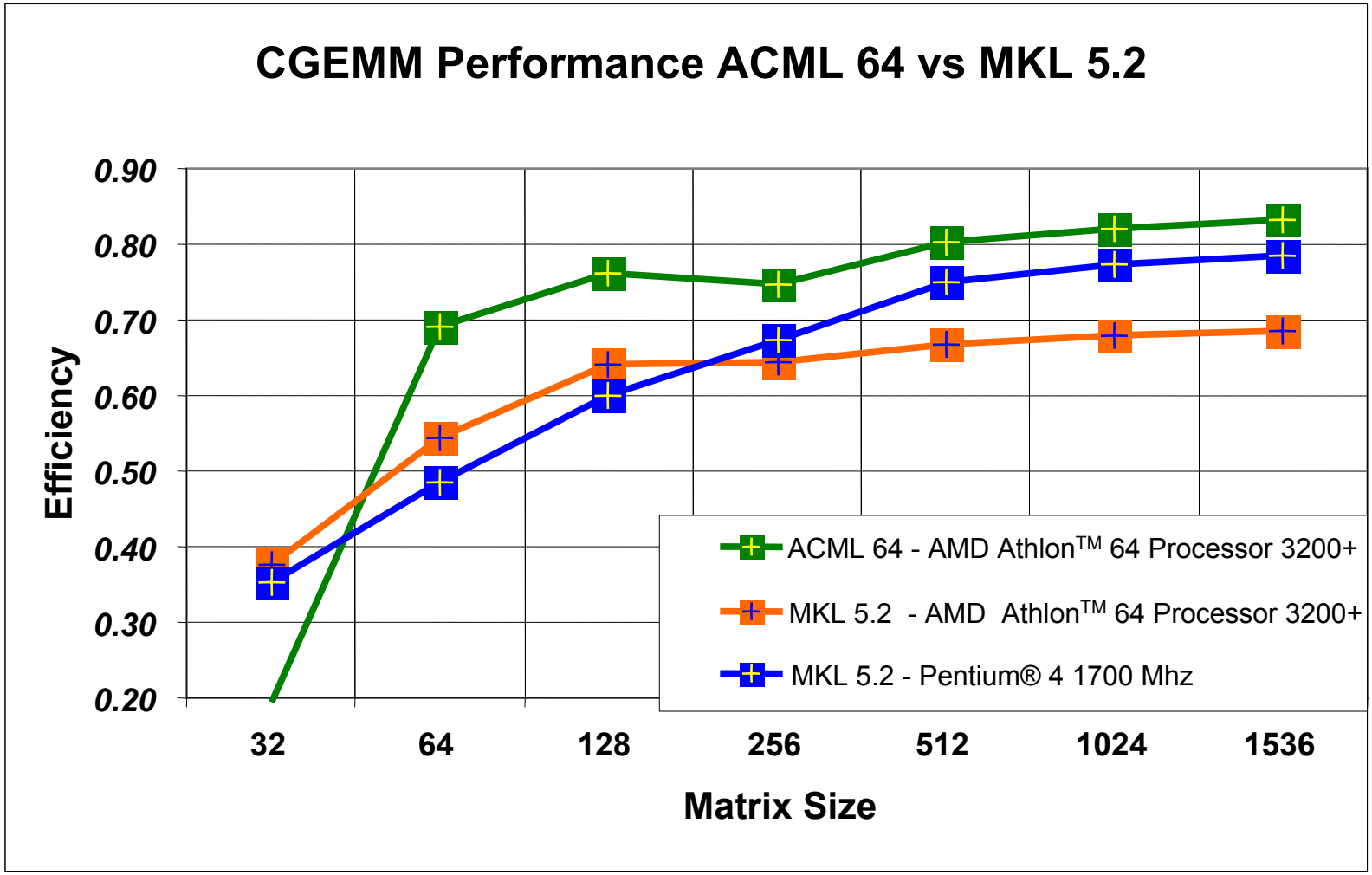
64-bit ACML v1.5 BLAS Performance

ZGEMM



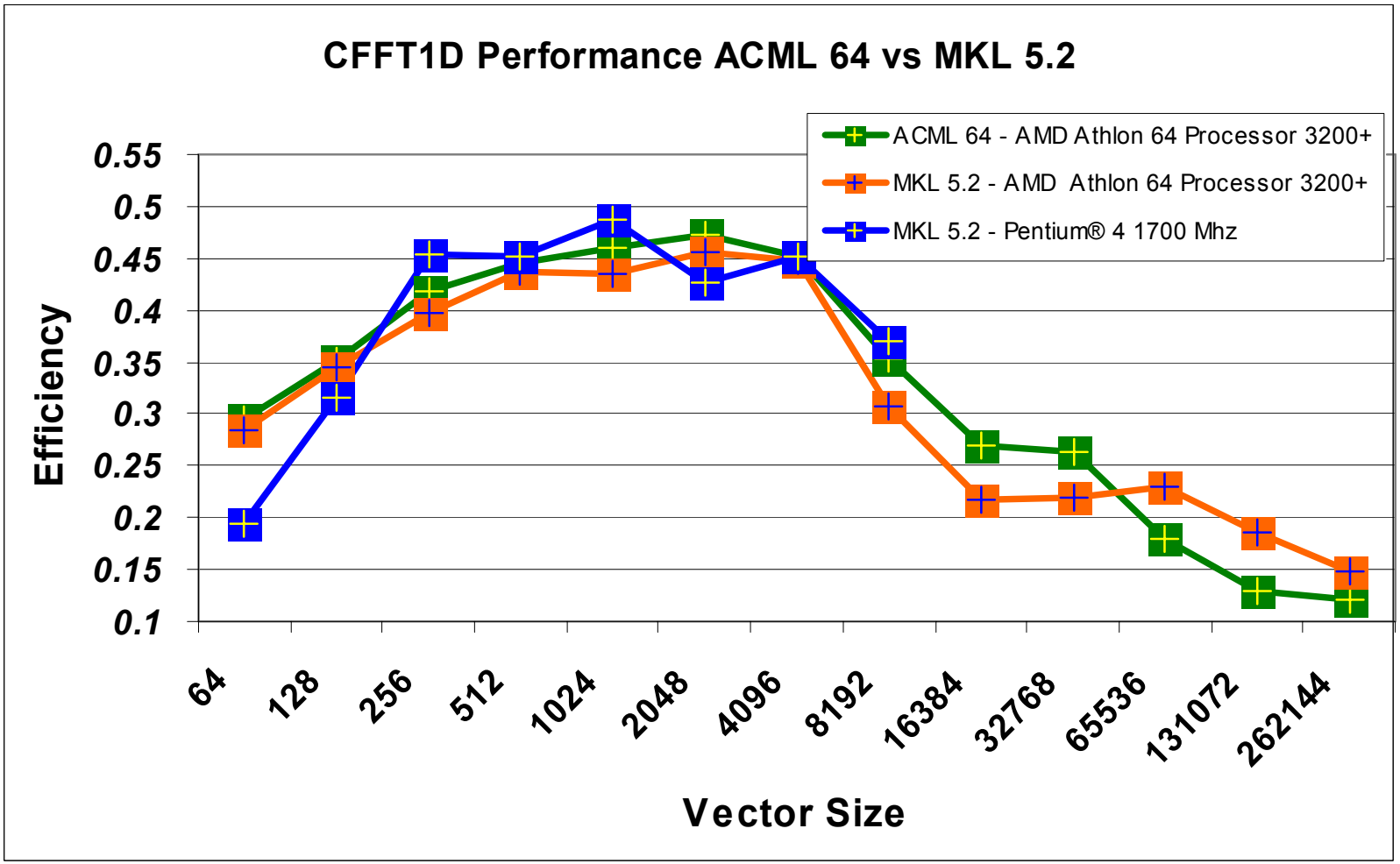
64-bit ACML v1.5 BLAS Performance

CGEMM



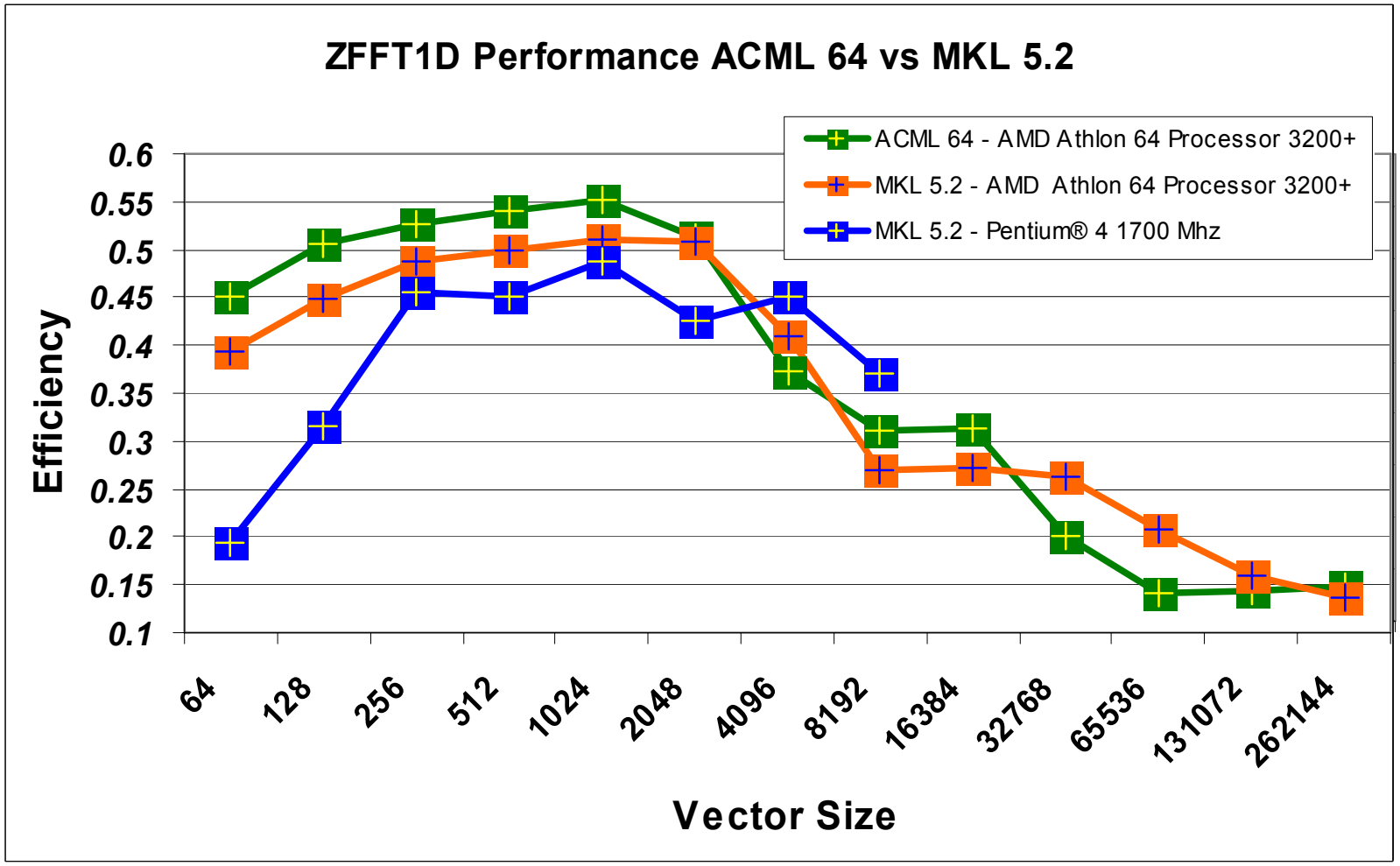
64-bit ACML v1.5 FFT Performance

CFFT1D



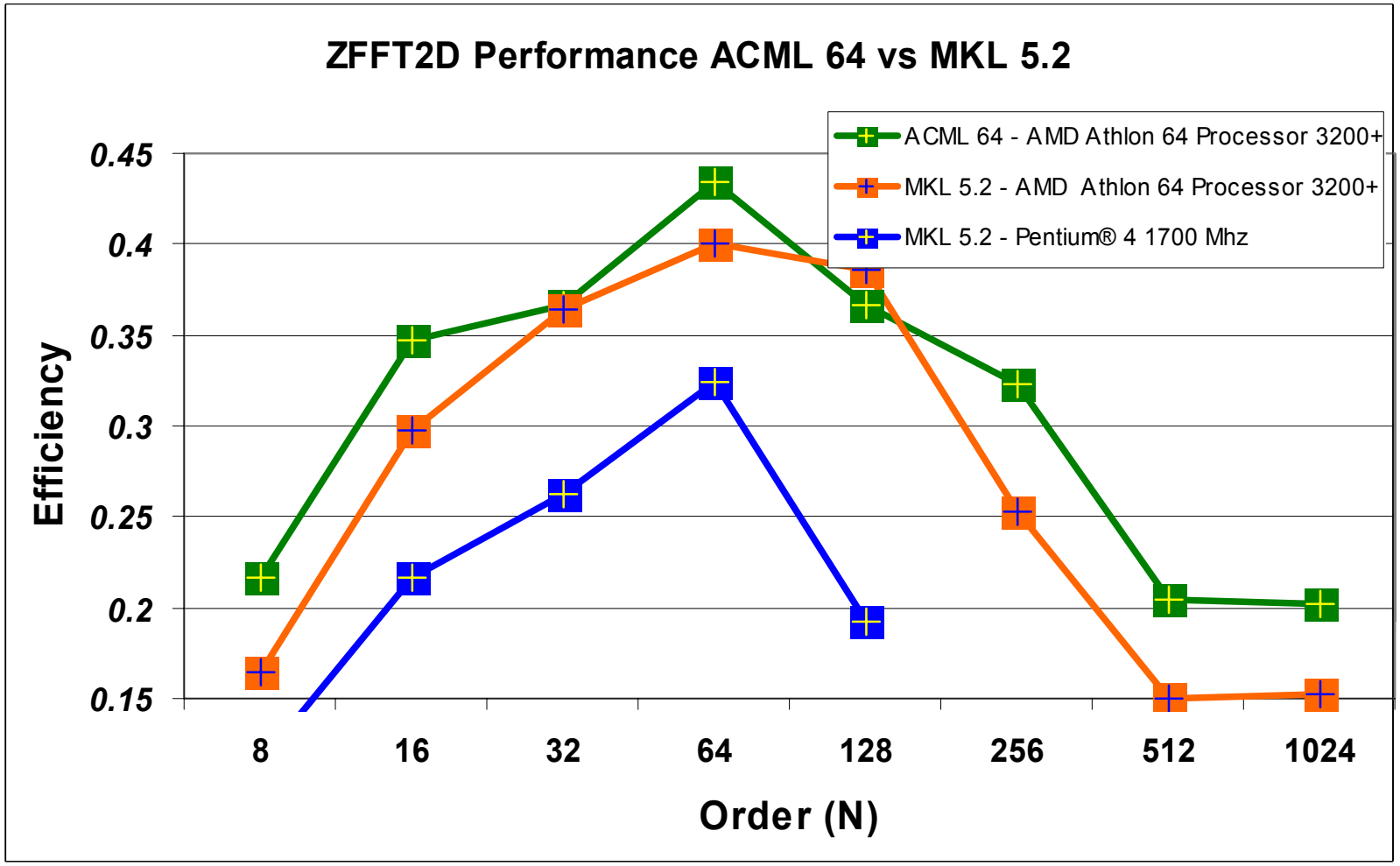
64-bit ACML v1.5 FFT Performance

ZFFT1D



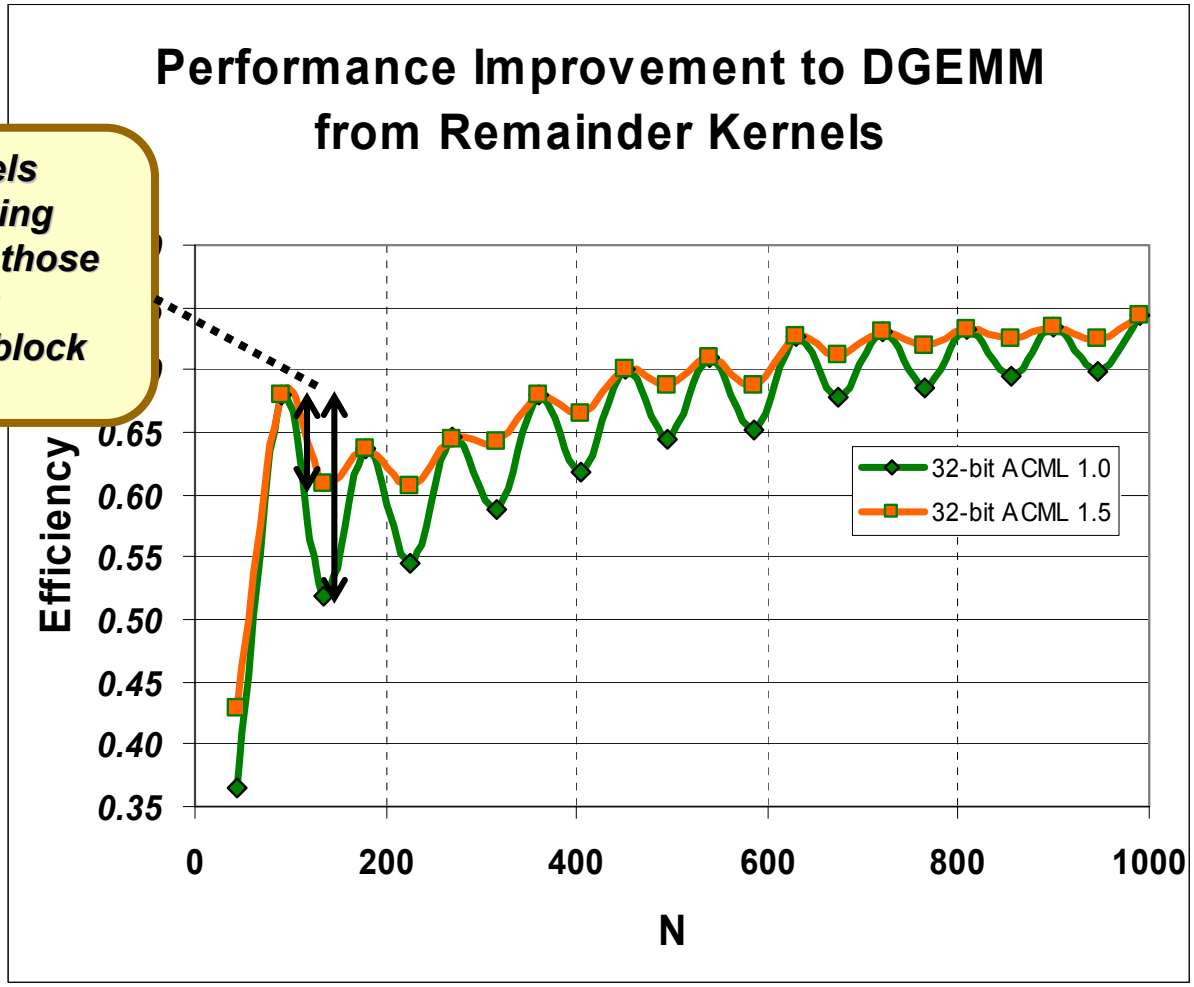
64-bit ACML v1.5 FFT Performance

ZFFT2D



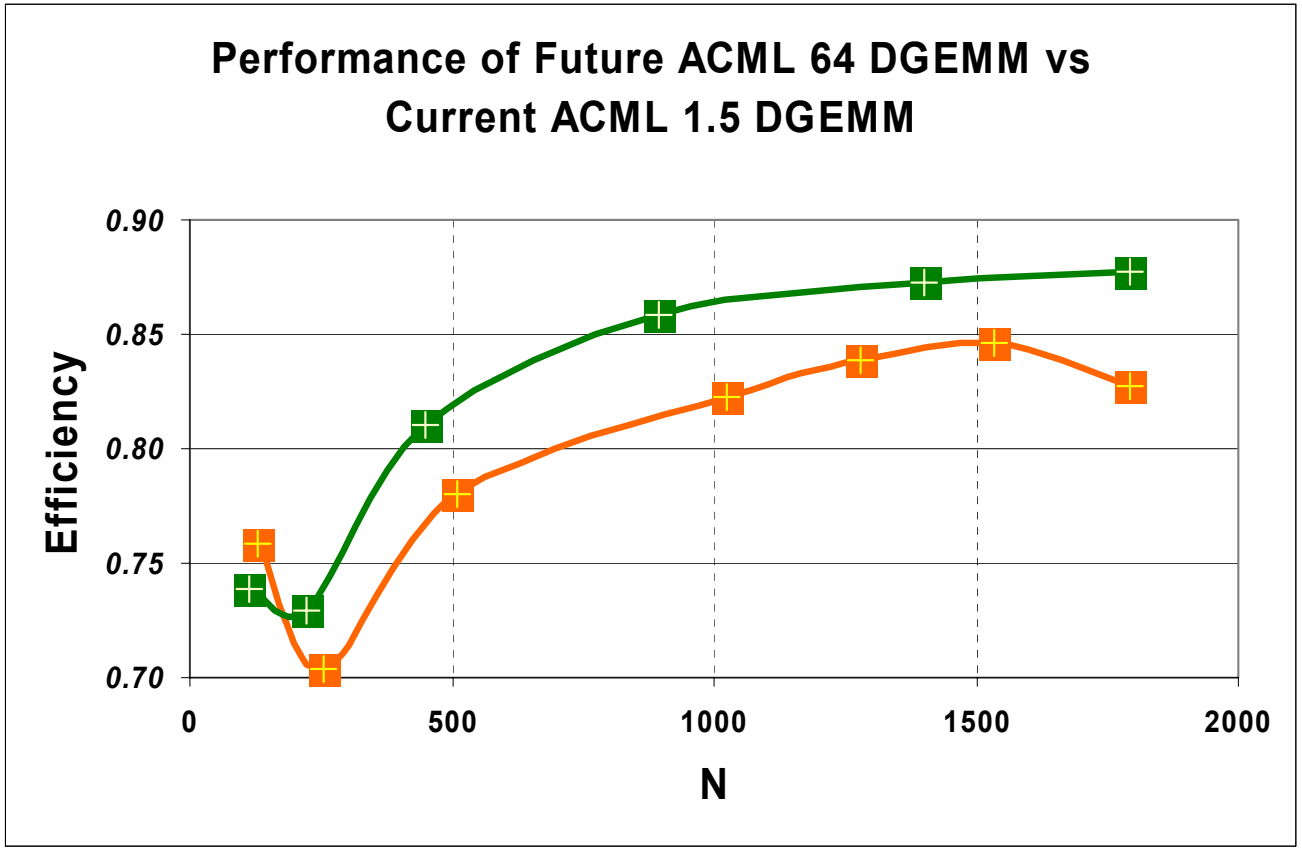
ACML v1.5 versus v1.0
32-bit BLAS Improvements

Remainder kernels assist in preserving performance for those cases that aren't multiples of the block size



Future BLAS Improvements
Example: DGEMM

- Smaller Block Size
  - *Less time spent in remainder kernels*
- Transposed Kernels
  - *Accumulation of registers eliminated*
- Minimized Bank Conflicts
  - *Increased frequency of 2 loads / cycle*
- 4-Byte offset loads
  - *Less pointer arithmetic*

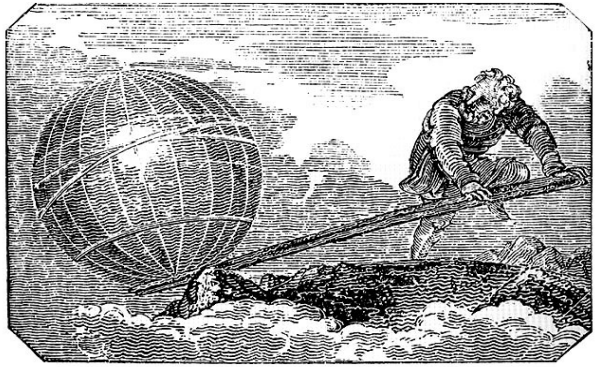


Conclusion and Closing Points

- **How good is our performance?**
  - *Averaging over 70 BLAS/LAPACK/FFT routines*
  - *Computation weighted average*
  - *All measurements performed on an* AMD Athlon™ 64 3200 processor

ACML 32-bit is **65% faster** than **MKL 5.2**

ACML 64-bit is **76% faster** than **MKL 5.2**



"Give me a long lever and a place upon which to stand and I will move the world " Archimedes circa 250 B.C

Benchmark System Configuration

AMD Platform

- **Processor**: *AMD Athlon™ 64 Processor 3200+*
- **Motherboard**: *1P Solo AMD Development Platform*
- **Hard Drive**: *40 GB IDE* 7200 RPM
- **Memory**: *2 x DDR 2100 – 512 MB*
- **OS**: *AMD64 SUSE SLES-8 RC7 2.4.19#1*

Intel Platform

- **Processor**: *Intel Pentium® 4 Processor 1.7 Ghz*
- **Motherboard**: *Intel D850GB*
- **Hard Drive**: *40 GB IDE* 7200 RPM
- **Memory**: *2 x 400 Mhz RDRAM – 512 MB*
- **OS**: *Red Hat Linux 9 (Shrike) / Kernel Revision 2.4.20-8*

Trademark Attribution